Editing algorithms for texts over formal grammars

F. Schwarzenberger & H. Zemanek

Economic automation is based on frequent repetition and clear rules. The principle is to analyse a process and to find out those parts which can be defined by a set of rules and where the repetition rates invite mechanical execution; high repetition rates can be gained by splitting the process into elementary steps. Most human activities can be automated economically, but it is true that so far we know only in a few cases how to do it. In the case of typesetting, the editing procedures are an excellent example of the application of the principle of automation. There are features of artistic or at least quasi-artistic character, that totally elude logical description. But there are a lot of mechanical rules which a computer can follow much better than a human being and which are to be applied so frequently that the computer can offer the more economic solution. We can expect order and clearness, precision and regularity from automation, but we should not expect the tastefulness a typesetting artist could add to the picture.

We are at the beginning of the algorithmic century in typesetting where nobody can offer complete solutions. One can look around and find and apply those methods which have been useful elsewhere and one can try to apply them to the editing problem. Since the Vienna Laboratory of IBM is concerned with the syntax and the semantics of formal languages, it was inviting to apply the principles of syntax analysis to typesetting, and editing in particular.

There exist numerous texts which have to be edited frequently and which follow tacitly more or less rigorous syntactical rules. Bibliographical records are a good example and stand for many similar texts. Already in the time before automation, the human editors intuitively have simplified their job and have made it more reliable by introducing certain writing rules. An automatic system has to make use of this situation, ie of the structural information inherent in the received text to be processed (the source text), for performing the same editing changes the human editor and the typesetter would apply in order to achieve the final form, the target text. The method we propose is based on syntax-directed techniques as developed for programming languages, and we try to show that editing can be advantageously handled by this method for a large class of texts. An experimental system based on this method is planned.

In the proposed system the editing process is performed in three steps:

1 Parsing of the source text (production of tree structure)
2 Modification of the text structure
3 Formatting of the target text

The block diagram of the system is shown in Fig 1. The tree structure of the source text is produced by a program which is provided with the set of syntactical rules of the source texts. The modification is based on the analysis of the source text, so that structure and content of the input text are represented in tree form. The tree, and thereby the structure of the text, can be remodelled by modification statements while its configuration on the printing sheet is controlled by format statements.

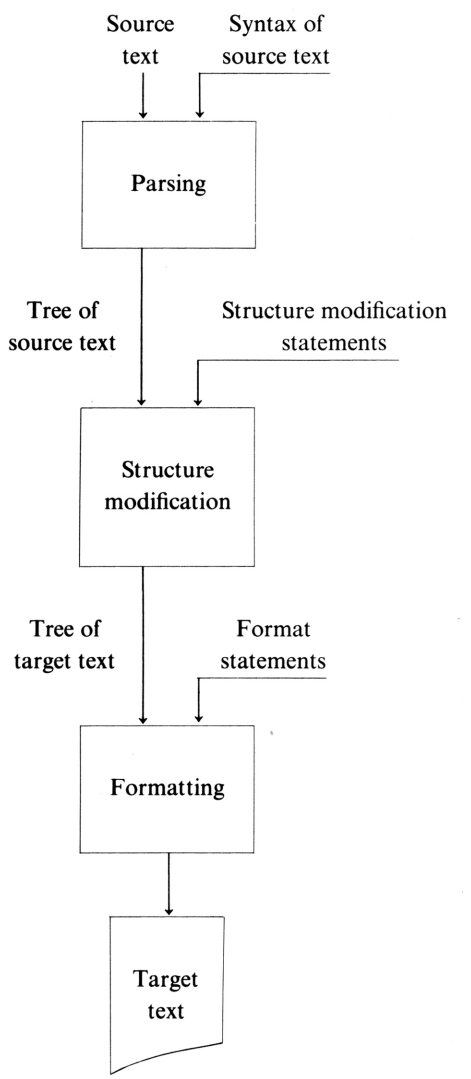


Fig 1 Syntax-directed editing system

The advantage of the proposed system is that instead of supplying edit instructions repeatedly at specific points in the text only

general rules with respect to the structure are supplied. The machine will then derive automatically any necessary action at the specific points in the text.

Source texts

The proposed editing system processes texts over formal grammars, ie texts the syntax of which can be formally described in an appropriate metalanguage. Such restrictions are not as rare in daily life as one might think. They are not noticed because usually the metalanguage is left to the insight of the reader instead of being formally declared. As an example, bibliographical records usually are written to conform to certain rules requesting spaces, new lines, punctuation marks or other separators, exactly for safeguarding their structure.

Now a main area of application visualized is scientific publications which are essentially based on a large number of formulae, such as in the fields of formal logic, mathematics and publication of algorithms. Especially in the case of published algorithms, which in most cases have been tested, it is critical to risk errors originating from the edit process after the program has been debugged. On the other hand, it is commonly found that programs are not properly arranged for their final printed version but require a heavy edit process. The proposed system seems to guarantee to a high degree the required edit process. In the case of published algorithms, the text is not only intended for the human reader but also for a computer.

Parsing of the source text

The source texts can be decomposed syntactically by means of automatic parsing algorithms, provided that the syntactic rules governing the source texts have been specified to the computer. The decomposition process provides as its result the structure of each source text in tree form. The tree structures are stored in the computer memory for eventual further processing.

In this paper, we use as metalanguage—the Backus notation—for expressing the syntax of the source texts. The Backus notation as a formalism for syntactic description is defined by the example:

$$\langle ab\rangle ::= (|[|\langle ab\rangle(|\langle ab\rangle\langle d\rangle$$

Sequences of characters enclosed in the brackets $\langle\ \rangle$ represent metalinguistic variables whose values are sequences of symbols. The marks $::=$ and $|$ are metalinguistic connectives. The mark $|$ means *or*. Any mark in a formula, which is not a variable or a connective, denotes itself (or the class of marks which are similar to it). Juxtaposition of marks and/or variables in a formula signifies juxtaposition of the sequences denoted.

By syntax we always mean in the following a generative syntax, that is, a set of rules by which syntactically correct texts can be generated. A parsing algorithm based on such a syntax then yields for any given text a reconstruction of those rules by which the text can be generated and in which order these rules are to be applied. This result can most conveniently be represented in tree form where each intermediate node of a tree is associated with a specific syntax rule and the terminal nodes are associated with the respective basic characters of the source text. Each syntactical rule defines a syntactical category, ie the set of phrases that can be generated by the syntactical rule. One may, therefore, also say that each intermediate node is associated with a certain syntactical category.

As a simple example, Fig 2 shows a set of syntax rules together with a section of the corresponding tree structure.

Syntax rules

⟨author⟩ : : = ⟨surname⟩, ⟨first names⟩
⟨surname⟩ : : = ⟨letter string⟩
⟨first names⟩ : : = ⟨first name⟩|⟨first name⟩b⟨first names⟩|
|⟨first name⟩b⟨initials⟩|⟨initials⟩
⟨first name⟩ : : = ⟨letter string⟩
⟨letter string⟩ : : = ⟨letter⟩|⟨letter⟩⟨letter string⟩
⟨initials⟩ : : = ⟨letter⟩.|⟨letter⟩.⟨initials⟩
⟨letter⟩ : : = a|b|c|d . . .|z|
A|B|C|D . . .|Z|

Tree

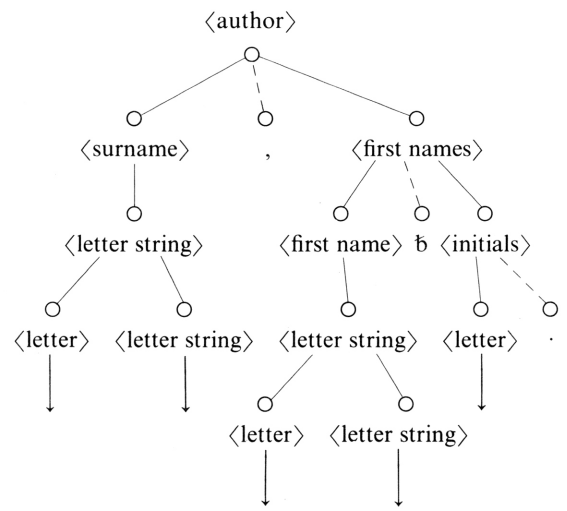


Fig 2 Syntax rules and tree

A fair number of algorithms and programs for automated syntactical analysis has become known and can be adopted for the present investigation in a suitable way. An essential requirement for syntax decomposition programs is that the decomposition process should be performed at a reasonable speed. For efficient parsing systems the decomposition speed should be at a rate comparable to the processing speed of input/output equipment.

Modification of the text structure

All changes and alterations which are to be applied to the source text for achieving the desired target text can be performed by proper remodelling of the source text tree into the tree representation of the target text.

The advantage of the proposed system is that instead of supplying edit instructions repeatedly at specific points in the text only general rules with respect to the structure are supplied. The machine will then derive automatically any necessary action at the specific points in the text.

For the present investigation we used a subset of the tools provided in syntax-directed compilation as described by P. Z. Ingerman. The structure modification may be described by describing the effect of the desired changes with respect to each individual syntax rule of the source text. In order to refer to each component (ie one syntactic category) of an alternative of a specific rule, these components (except the terminals) are numbered from left to right beginning with 1. Each component number is placed into square brackets. As a simple example, in Rule No. 21 of Example 1 the components would be numbered as follows:

21 ⟨source⟩ : : = ⟨place⟩ : ⟨publisher⟩, ⟨year⟩
[1] [2] [3]

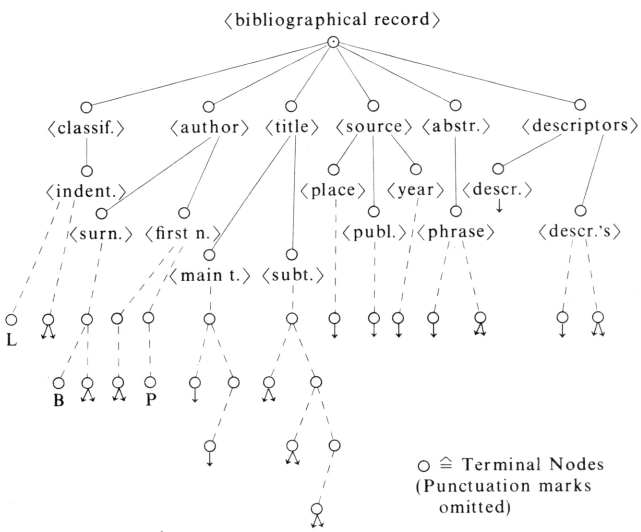
The desired structural modification can now be stated by specifying the reordering (and omission) of the components and the insertion of the desired punctuation marks. With respect to the above example, interchange of ⟨place⟩ and ⟨publisher⟩ and replacement of ':' by ',' and replacement of ',' by 'b' can simply be specified by the following structure modification statement:

21 [2], [1] b [3]

The above statement specifies uniquely the change of the tree representation of the text and specifies also the new syntax rule of the modified text. The new syntax rule of the above example obviously would read

⟨source⟩ : : = ⟨publisher⟩, ⟨place⟩ b ⟨year⟩

Tree of source text



Tree of target text

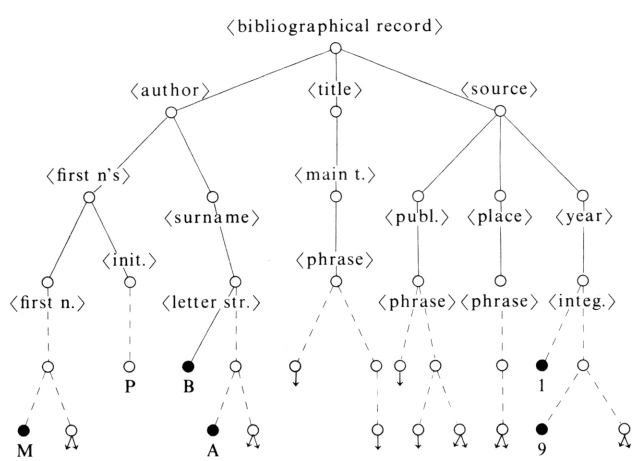


Fig 3 Tree representations (with respect to Example 1)

Formatting

In contrast to natural language text where the formatting process in most cases is sufficiently limited to hyphenation, justification and indention, the rules for separation and configuration of highly structured text (eg of mathematical formulae, texts of formal logic, programs) are much more complex. Exactly in the latter case it is necessary to support the appearance of the structure of the text by a proper physical arrangement. It seems, therefore, appropriate to make the system accept format statements which explicitly refer to the structure of the text, ie the syntactical description.

In the following some of the formatting tasks which can be accomplished by means of the proposed method are illustrated by means of some format statements used in the examples. A given set of format statements together with the tree of the unformatted target text controls the print process and specifies the appearance of text on a page.

Format statements

For the sake of simplicity, it is assumed that the target text will be printed onto rectangular pages with equidistant printing and line positions. It is also assumed that the text will appear on a page running left to right and from top to bottom. The format statements used in the present study are composed as follows:

NAME (format parameter, qualification, (argument list)),

where Name denotes a specific format statement,
Format parameter specifies indentions,
Qualification specifies the context (scope) in which the format statement is to be applied, and
Argument list specifies those syntactic categories to which the format statement refers.

Note that an arbitrary set of format statements does not necessarily guarantee that there exists a printed page according to the

specifications. Format parameters such as: line width (ie maximum number of printing positions per line); page depth (ie maximum number of lines per page); absolute left margin (ie first admissible printing; position on a line); and head line (ie first admissible line position on a page), are to be specified as initial constants during the formatting phase.

Separation

For properly configurating the target texts on given physical lines and pages, rules for separation are required. It should be possible to state between which syntactical entities NEW LINE or NEW PAGE must or may be started, and under which circumstances these rules are to be applied.

Format statement for line separation: SEPLINE (Qualification, (argument list)), where argument list specifies those syntactic categories of target text for which a new line is to be started if line overflow would occur.

With regard to an admitted line width, in case of line overflow line separation may be performed at any point of text with respect to the specifications in the argument list of SEPLINE. It actually is performed at the separation point closest to the right margin. If not specified otherwise, the separated text part is continued in the next line according to the valid left margin. The valid left margin follows from indention statements (see below) as applied to a certain qualified syntactic category with respect to the previous valid left margin. The initial value of the valid left margin is equal to the value of the absolute left margin.

Format statement for page separation: SEPPAGE (qualification, (argument list)), where argument list specifies those syntactic categories of target text for which a new page is to be started if page overflow would occur.

With regard to an admitted page depth, in case of page overflow page separation may be performed at any point of text with respect to the specifications in the argument list of SEPPAGE. It actually is performed at the separation point closest to the page bottom. If not specified otherwise, the separated text part is continued on the next page in the first line according to the valid left margin.

Format statement for start of new line: NEWLINE (qualification, (argument list)), where argument list specifies those syntactic categories of target text for which a new line is to be started unconditionally with respect to the valid left margin.

Format statement for start of new page: NEWPAGE (qualification, (argument list)), where argument list specifies those syntactic categories of target text for which a new page is to be started unconditionally.

Indention

A frequently used way to make the structure of text apparent is indention. Two types of indention are distinguished, namely relative and absolute indention.

Format statement for absolute indention: ABSINDENT (t_{AI}, qualification, (argument list)), where t_{AI} denotes the number of indentions with respect to the absolute left margin and argument list specifies those syntactic categories of target text for which a new line should be started and which are to be indented by t_{AI} printing positions with respect to the absolute left margin. Valid left margin : = t_{AI}.

Format statement for relative indention: RELINDENT (t_{RI}, qualification, (argument list)), where t_{RI} denotes the number of indentions relative to the valid left margin and argument list specifies those syntactic categories of target text for which a new left margin is to be set by indenting the text by t_{RI} printing positions relative to the valid left margin. Valid left margin : = previous valid left margin + t_{RI}. Note that RELINDENT defines a hierarchy of indentions in case a syntactic category contained in the argument list is self-embedding.

Adjustment

A slight generalization of indention leads to the frequently used configuration where an optional syntactically identifiable point of a syntactic category is to be placed at a specific tabulator position. This process is called adjustment. Similar to indention one can distinguish between absolute and relative adjustment.

Tables and forms

In the case where a given target text is to be presented in form of a table it again seems natural to refer to the structural description of the text. It is assumed that the text is a list of uniformly structured entities. The text is to be presented in form of a table such that each of the entities is one entry. The layout of the table (vertical lines on a page) and the assignment of syntactical categories to particular columns must be specified to the machine. Each of the columns now can be regarded as one page, ie rules for separation, indention and adjustment can be applied accordingly. Forms can be viewed as a generalization of the above case of tables. While in the latter case the page has been divided into columns, the page is now assumed to be partitioned into rectangular fields. The partitioning and the assignment of syntactical categories to particular fields must be specified to the machine to define the format process. Again, each particular subfield can be conceived as a single page. Thus separation, indention and adjustment can be applied accordingly.

The most general organization considered in the present study is the following system. One starts to specify printing blocks for substructures of the given target text whereby the shape of the printing block may depend on the syntactical categories of the substructure. The arrangement of the printing blocks is then specified in dependence of the embedding structure with respect to a logical page. The last step is the breakdown of logical into physical pages. Note that all formatting processes previously

mentioned can be applied to each printing block accordingly. Production of flow charts from programs in linear form may serve as a special example.

Example 1: Bibliographical records

Syntax of source text (library card)

Syntax rule No.

```
1  ⟨bibl. record⟩ : : = ⟨classification⟩/⟨authors⟩/⟨title⟩/
                        ⟨source⟩/⟨abstract⟩/⟨descriptors⟩//
2    ⟨classification⟩ : : = ⟨identifier⟩
3      ⟨identifier⟩ : : = ⟨letter⟩ | ⟨letter⟩⟨identifier⟩ |
                          ⟨digit⟩⟨identifier⟩
4        ⟨letter⟩ : : = a | b | c | d ... |z|
                        A | B | C | D ... |Z|
5        ⟨digit⟩ : : = 0 | 1 | 2 | 3 ... |9|
6    ⟨authors⟩ : : = ⟨author⟩ | ⟨author⟩ ; ⟨authors⟩
7      ⟨author⟩ : : = ⟨surname⟩, ⟨first names⟩
8        ⟨surname⟩ : : = ⟨letter string⟩
9        ⟨first names⟩ : : = ⟨first name⟩ | ⟨first name⟩ ␣
                             ⟨first names⟩ | ⟨first name⟩ ␣
                             ⟨initials⟩ | ⟨initials⟩
10         ⟨first name⟩ : : = ⟨letter string⟩
11           ⟨letter string⟩ : : = ⟨letter⟩ | ⟨letter⟩⟨letter
                                   string⟩
12         ⟨initials⟩ : : = ⟨letter⟩ . | ⟨letter⟩ . ⟨initials⟩
13   ⟨title⟩ : : = ⟨main title⟩ | ⟨main title⟩ * ⟨subtitle⟩
14     ⟨main title⟩ : : = ⟨phrase⟩
15     ⟨subtitle⟩ : : = ⟨phrase⟩
16       ⟨phrase⟩ : : = ⟨word⟩ | ⟨word⟩ ␣ ⟨phrase⟩
17         ⟨word⟩ : : = ⟨identifier⟩ | ⟨integer⟩ | ⟨special
                        character⟩
18           ⟨identifier⟩ : : = ⟨letter⟩ | ⟨letter⟩⟨identifier⟩ |
                                ⟨digit⟩⟨identifier⟩
19           ⟨integer⟩ : : = ⟨digit⟩ | ⟨digit⟩⟨integer⟩
20           ⟨special character⟩ : : = . | , | —
21   ⟨source⟩ : : = ⟨place⟩ : ⟨publisher⟩, ⟨year⟩
22     ⟨place⟩ : : = ⟨phrase⟩
23     ⟨publisher⟩ : : = ⟨phrase⟩
24     ⟨year⟩ : : = ⟨integer⟩
25   ⟨abstract⟩ : : = ⟨phrase⟩
26   ⟨descriptors⟩ : : = ⟨descriptor⟩ | ⟨descriptor⟩ ;
                         ⟨descriptors⟩
27     ⟨descriptor⟩ : : = ⟨phrase⟩
```

Structure modification statements

applied to Syntax Rule No.

```
1        [2] : [3] . – [4]
7        [2] ␣ [1]
13       [1] | [1]
21       [2], [1] ␣ [3]
```

Format statements

```
SEPLINE (⟨bibliographical record⟩, (⟨author⟩))
SEPLINE (⟨title⟩, (⟨word⟩))
SEPLINE (⟨source⟩, (⟨word⟩))
NEWLINE (⟨bibliographical record⟩, (⟨author⟩))
NEWLINE (⟨bibliographical record⟩, (⟨title⟩))
NEWLINE (⟨bibliographical record⟩, (⟨source⟩))
ABSINDENT (3, ⟨bibliographical record⟩, (⟨title⟩))
ABSINDENT (6, ⟨bibliographical record⟩, (⟨source⟩))
```

Source text

LC 6527789/BARNETT, Michael P./Computer Typesetting* Experiments and Prospects/Cambridge: The M.I.T. Press, 1965/Describes in detail the experimental work performed by the author and his colleagues at M.I.T. The technical descriptions of the equipment, the programming tactics, explanation of traditional printing methods, basic data on film-setting and syntactic analysis are presented in a manner which should be intelligible to those with either a printing or data processing background. Some of the implications of this work to the printing industry are discussed./Computer Application; Typesetting; Editing Programs; Photon//

Unformatted target text (See Fig 3)

Michael P. BARNETT: Computer Typesetting.—The M.I.T. Press, Cambridge 1965.

Formatted target text

Michael P. BARNETT:
Computer Typesetting.—
The MIT Press, Cambridge 1965.

Example 2: ALGOL program

This example shows how the structure of an ALGOL program can be made apparent by applying format statements with respect to the Algol syntax for achieving a proper two-dimensional representation. Obviously, similar rules for configurating program texts are observed by many authors and editors.

Unformatted target text: linear representation of Algol program

```
begin integer d, i, j, k, r; real t; d: = gcd(m, n);
r: = (m + n) ÷ d; for i: = 1 step 1 until d do begin
j: = i; t: = a[i]; for k: = 1 step 1 until r do begin
if j = m then j: = j + n else j: = j − m; swap(t, a[j])
end k end i end
```

Syntax of target text (modified Algol-syntax)

```
⟨program⟩ : : = ⟨block⟩
  ⟨block⟩ : : = begin ⟨body⟩ end
    ⟨body⟩ : : = ⟨declaration list⟩; ⟨statement list⟩
      ⟨declaration list⟩ : : = ⟨declaration⟩|
                               |⟨declaration⟩; ⟨declaration list⟩
        ⟨declaration⟩ : : = ...**
      ⟨statement list⟩ : : = ⟨statement⟩|
                             |⟨statement⟩; ⟨statement list⟩
        ⟨statement⟩ : : = ⟨for statement⟩|⟨block⟩|
                          |⟨conditional statement⟩ ...**
          ⟨for statement⟩ : : = ⟨for clause⟩⟨statement⟩
          ⟨conditional statement⟩ : : = ⟨if clause⟩|
                                        |⟨if clause⟩ else ⟨statement⟩
            ⟨for clause⟩ : : = ...**
            ⟨if clause⟩ : : = ...**
```

Format statements

```
SEPLINE (⟨body⟩, (⟨declaration⟩, ⟨statement⟩))
NEWLINE (⟨block⟩, (begin, ⟨declaration list⟩,
          ⟨statement list⟩, end))
  NEWLINE (⟨for statement⟩, (⟨for clause⟩))
  NEWLINE (⟨conditional statement⟩, (⟨if clause⟩))
RELINDENT (3, ⟨block⟩, (⟨body⟩))
  RELINDENT (5, ⟨conditional statement⟩, (then ⟨state-
              ment⟩, else ⟨statement⟩))
```

Formatted target text: structured representation of Algol program

```
begin
  integer d, i, j, k, r; real t;
  d: = gcd(m, n); r: = (m + n) ÷ d;
  for i: = 1 step 1 until d do
  begin
    j: = i; t: = a[i];
    for k: = 1 step 1 until r do
    begin
      if j = m
        then j: = j + n
        else j: = j − m
      swap(t, a[j])
    end k
  end i
end
```

In conclusion, some boundaries of the proposed system should be mentioned. Main fields of application could be in editing highly structured texts such as mathematical and logical formulae, programs, bibliographic records and catalogues. The suggested method will be less essential for editing running texts of natural languages as contained, eg in novels or work of fiction, due to the less distinct structural configuration of such texts.

Furthermore, features for performing local text corrections which are contained in many other editing systems are not considered in the proposed system. The efficiency at which syntax decomposition and structure modification can be performed is an important criterion for the practical impact of the proposed system.

Acknowledgement

We are indebted to Mr P. Lucas for many basic ideas, to Mr E. Neuhold for his co-operation, and to many other members of the Vienna laboratory for comments and suggestions.

During the last part of the work on the paper, we received the thesis of Mr J. M. Gerard from the Institute of Computer Sciences (London University) on *Synthesising nonlinear forms from linear descriptions by Syntactic analysis.* This paper describes another approach.

Literature

Backus, J. *The syntax and semantics of the proposed international algebraic language* of the Zurich ACM-GAMM Conference. Proc. ICIP Conference, Paris, 1959. UNESCO Paris—Oldenbourg Munich—Butterworth London 1960.

Barnett, M. P. *Computer typesetting—experiments and prospects.* The MIT Press, Cambridge & London. 1965.

Brooker, R. A. & Morris, D.—A general translation program for phrase structure languages.—*J. ACM* 9 1962, No. 1.

Chomsky, N. On certain formal properties of grammars. *Information and Control* **2,** 1959, No. 2.

Eickel, J., Paul, M., Bauer, F. L. & Samelson, K.—A syntax-controlled generator of formal language processors.—Comm. ACM 6 (1963) No. 8, pp. 451–455.

Gerard, J. M. Synthesising non-linear forms from linear descriptions by syntactic analysis. Ph.D. Dissertation, University of London 1965 (unpublished).

Ingerman, P. Z. A syntax-oriented compiler for languages whose syntax is expressible in Backus normal form, and some proposed extensions thereto. Master's Thesis, Moore School of Electrical Engineering, University of Pennsylvania, 1963.

Irons, E. T.: The structure and use of the syntax-directed compiler, in: *Annual Review in Automatic Programming*, **3** Pergamon Press, New York and London 1963.

Knuth, D. E.: On the translation of languages from left to right. *Information and Control* **8** 1965, No. 6.

Naur, P. (ed.): Revised report on the algorithmic language ALGOL 60—*Comm. ACM* **6,** 1963, No. 1.